# Utilisation d'un système de construction logicielle

Les systèmes de construction logicielle, *build systems* en anglais, plus connus sous l'expression *moteur de production*, sont des logiciels dont l'objet principal est d'automatiser, directement ou indirectement, le processus de compilation d'un code source afin de faciliter le test et la distribution des produits logiciels.

Le moteur de production **CMake**, populaire du fait de sa caractéristique *multi-plateforme*, utilise essentiellement des fichiers de configuration, nommés `CMakeLists.txt`, pour contrôler le processus de compilation. Toutefois, CMake ne génère pas des exécutables mais des fichiers de constructions standards en fonction de la plateforme. Par exemple, une même configuration CMake génèrera des fichiers `Makefile` sous Linux et des fichiers de projet Visual Studio sous Windows.

Un fichier `CMakeLists.txt` contient une liste d'appels à des commandes **CMake**. Un appel à une commande suit la syntaxe suivante : `<nom-commande> (<arg1> <arg2> ...)`. À titre d'exemple, la présence de la commande `cmake_minimum_required (VERSION 3.3)` indique qu'il faut utiliser une version de **CMake** postérieure à la version 3.3. Référez-vous à la documentation de CMake pour une liste des commandes :
https://cmake.org/cmake/help/v3.3/manual/cmake-commands.7.html

## Récupérer un projet utilisant cmake

Exercice 1 – Placez-vous dans un répertoire temporaire, puis exécutez la commande suivante pour récupérer les sources du projet *gamefromrussia* : `svn export http://gamefromrussia.googlecode.com/svn/trunk/`

Compilez le projet et lancez l'exécutable généré. Regardez le contenu du fichier `CMakeLists.txt`

## Premiers pas avec CMake

Pour votre première expérience avec CMake, vous serez amenés écrire les fichiers de configuration nécessaires à la compilation du projet `magasin` ayant servi de support au TD sur `make`. Récupérez et décompressez l'archive `fichiers-02.tar.gz`. L'arborescence a la structure suivante (identique à celle mise en place lors des premiers TD) :

```
bcb/            # le répertoire contenant les sous-répertoires memoire/ et chaine/
sdd/            # le répertoire contenant les sous-répertoires paire/, vext/ et file/
magasin/        # le répertoire contenant le code du programme principal utilisant
                # les fonctions dont le code est défini dans sdd/ et bcb/
```

À tout moment, vous pouvez tester si vos fichiers CMake sont corrects en invoquant créant un répertoire `build` au même niveau que votre fichier `CMakeLists.txt` racine, puis en exécutant depuis ce répertoire la commande `cmake ..` (qui doit s'exécuter normalement si la syntaxe de vos fichiers CMake est correcte) suivie de `make` (qui doit s'exécuter sans erreur si vos instructions CMake décrivent un projet cohérent). La commande `make` doit produire des fichiers à partir de la question 4, et les variantes `make install` et `make check` permettent de tester vos réponses aux questions 8 et 9 respectivement.

Exercice 2 – À la racine du projet, c'est-à-dire dans le répertoire contenant `sdd`, `bcb` et `magasin`, créez votre premier fichier de configuration `CMakeLists.txt` où vous indiquerez la version minimum requise de CMake (`cmake_minimum_required`) et le nom du projet (`project`).

Exercice 3 – À l'aide de la commande `add_definitions`, indiquez à `CMake` que le compilateur `gcc` doit considérer que le langage de base est celui du standard `c99`.

Exercice 4 – Indiquez à CMake les répertoires où se trouvent les fichiers en-tête `.h` qui seront nécessaires à la compilation de votre projet (`include_directories`).

Finalement, indiquez les sous-répertoires du projet contenant les codes sources. Pour ce faire, utilisez la commande (`add_subdirectory`).

Exercice 5 – Allez dans le répertoire `bcb` et créez un fichier de configuration CMake. Ce nouveau fichier héritant des attributs définis dans le fichier de configuration du répertoire parent, il suffit d'y instruire l'ajout de la bibliothèque `bcb` grâce à la commande `add_library`, en indiquant le nom à donner à la librairie suivi des fichiers source à compiler pour produire cette bibliothèque.

Exercice 6 – Faites de même pour le répertoire `sdd`.

Exercice 7 – Allez dans le répertoire *magasin* et créez un fichier de configuration *CMake*. Spécifiez le nom de l'exécutable à produire ainsi que les fichiers source à compiler pour obtenir cet exécutable (*add_executable*) ; il vous faudra préciser séparément les noms des bibliothèques à lier à l'exécutable (*target_link_libraries*).

Exercice 8 – Une fois toutes les configurations écrites, retournez à la racine du projet et créez un répertoire *build/* (si ce n'est pas déjà fait). Déplacez-vous dans ce répertoire et exécutez la commande *cmake ..* pour générer les fichiers *Makefile*. Exécutez ensuite *make* directement dans le répertoire *build/* pour compiler. Une fois la compilation effectuée, vous devriez retrouver l'exécutable dans le répertoire *build/magasin/*.

Exercice 9 – Installation : faites les ajouts et modifications nécessaires des fichiers *CMakeLists.txt* de sorte que que l'exécutable *magasin* soit recopié dans le répertoire *bin/*, les bibliothèques dans *lib/* et les fichiers *.h* dans *include/*. Comme vous n'avez (théoriquement) pas d'accès *root* sur les machines du CREMI, vous ne pouvez pas installer le logiciel magasin à la racine du système de fichier. Afin de tester vos changements, ajoutez dans le fichier de configuration racine la commande *set(CMAKE_INSTALL_PREFIX répertoire/de/votre/choix)* (vous devez évidemment disposer des droits en écriture sur le répertoire choisi) [1].

Testez vos modifications à l'aide de la commande *make install*.

Exercice 10 – Tests : faites les ajouts et modifications nécessaires des fichiers *CMakeLists.txt* de sorte que les programmes de test du module *vext* soient également créés et testés. En particulier on souhaiterait que l'exécution de la commande *make check* lance l'exécution de tous les tests par la commande *ctest*.

Attention : certains tests peuvent boucler. Pensez à donner un timeout à l'exécution de ces tests à l'aide de la commande CMake *set_tests_properties*.

1. Observez que si vous ajoutez le répertoire en question suivi de *bin* dans votre variable d'environnement *PATH* (par ex. avec *export PATH=$PATH:$HOME/votre/prefixe/choisi/bin*), vous pouvez invoquer la commande *magasin* comme n'importe quelle autre commande de GNU/Linux.